CS-119(a) – ICC-C Série 7

2024-04-09

Rappel Pour faire lire des valeurs d'un fichier texte à votre programme il suffit d'utiliser la redirection de l'entrée :

```
./exo < fichier.txt
```

Exo1

Qu'affiche ce code ? Réfléchissez-y sans le faire tourner dans un premier temps.

```c
#include <stdio.h>

int inspect(const int *tab, int size)
{
    int big = tab[0];
    for (int i = 1; i < size; i++)
    {
        if (big < tab[i])
        {
            big = tab[i];
        }
    }
    return big;
}

int main()
{
    int v[] = {12, 1, 70, -4, 122, 223, 1, 2, 1000};

    printf("%d %d\n", inspect(v, 4), inspect(v + 4, 5));
}
```

Solution de l'exercice 1

La fonction `inspect()` prend un pointeur vers le début d'un tableau d'entiers, ainsi que sa taille, et renvoie le plus grand élément de ce tableau.

Cette fonction est appelée d'abord avec le tableau `v` et une taille de `4`, donc elle retournera le plus grand élément parmi les 4 premiers, c'est à dire `70`. Ensuite, la fonction est appelée avec le pointeur `v+4`, c'est à dire le "sous-tableau" qui commence au cinquième élément de v, avec la taille `5`. Donc, ce deuxième appel retourne le plus grand élément parmi les 5 derniers éléments du tableau, c'est à dire `1000`.

Le code affiche

```
70 1000
```

Exo2

Qu'affiche ce code ? Réfléchissez-y sans le faire tourner dans un premier temps.

```c
#include <stdio.h>
#include <string.h>

void process(char* string)
{
    char* gnirts = string + strlen(string) - 1;
    while (gnirts - string > 0)
    {
        char c = string[0];
        string[0] = gnirts[0];
        gnirts[0] = c;

        string++;
        gnirts--;
    }
}

int main()
{
    char texte[] = "super star";
    process(texte);
    printf("%s\n", texte);
}
```

Solution de l'exercice 2

La fonction `process()` prend en paramètre un pointeur vers un tableau de `char` (chaîne de caractères). Elle initialise un deuxième pointeur `gnirts` qui prend

l'adresse du dernier caractère de la chaîne. Ensuite, tant que le premier pointeur est "plus petit" que le deuxième, on inverse les valeurs des caractères – le premier pointeur stocke la valeur qui se trouve dans le deuxième, et vice-versa. Ça a du sens de comparer des pointeurs dans ce contexte, car ils représentent des adresses des éléments de la chaîne de caractères. Avant de passer à l'itération suivante, on incrémente le premier pointeur (il avance dans la chaîne) et on décrémente le deuxième (il recule dans la chaîne). Quand les deux pointeurs se croisent, on sort de la boucle.

Ce code va tout simplement *inverser* la chaîne de caractères. Le code affiche

```
rats repus
```

c'est à dire la chaîne `"super star"` lue à l'envers.

Exo3 Parenthèses simples

Écrivez un programme qui lit depuis l'entrée standard une suite de parenthèses et qui décide si c'est une suite "bien formée". La suite contient uniquement les caractères `'('` et `')'`. Dans une suite bien formée chaque parenthèse ouverte a une parenthèse fermée qui lui correspond et la suite de parenthèses comprise entre ces deux parenthèses est elle-même bien formée. La plus simple suite bien formée est la suite vide. Par exemple, les suites suivantes sont bien formées

```
()
(())
()(())(())
(())()()()((()))()())
```

mais ces autres suites ne le sont pas :

```
)
())
)(())
())(()
(())(
```

Indice En parcourant la suite depuis le début, à n'importe quel point le nombre de parenthèses ouvertes doit être toujours supérieur ou égal au nombre de parenthèses fermées, et à la fin de la suite les deux nombres doivent être égaux - on a autant de parenthèses ouvertes que fermées.

Solution de l'exercice 3

```c
#include <stdio.h>

int main()
{
    char expression[100];
    scanf("%s", expression);

    int compteur = 0;
    int ok = 1; // tout va bien
    for (char *p=expression; *p != 0; p++)
    {
        if (*p == '(')
        {
            compteur++;
        }
        else // ')'
        {
            compteur --;
        }
        if (compteur < 0)
        {
            // Trop de parenthèses fermées
            ok = 0;
            break;
        }
    }

    // Le compteur doit être 0,
    // sinon il y a des parenthèses ouvertes
    // qui n'ont pas été fermées
    if (ok && !compteur)
    {
        printf("Suite bien formée!\n");
    }
    else
    {
        printf("Suite mal formée :(\n");
    }
}
```

Exo4 Parenthèses complexes

Nous aimerions refaire l'exercice précédent, mais avec plusieurs types de parenthèses : les accolades et les crochets sont désormais aussi autorisées. Les caractères de la suite seront parmi les six suivants : `'{'`, `'}'`, `'['`, `']'`, `'('`, et `')'`.

Voici des exemples supplémentaires de suites bien formées :

```
[{[]()}(())]
[][]{}{()}
{}[[([])]{}]
```

et de suites mal-formées :

```
[(])
[]}{[]
[(})]
[[{{}]}]
```

Attention, la solution simple de l'exercice précédent risque de ne pas marcher, car pour la suite `"[{]}"` il y a toujours plus de parenthèses ouvertes que fermées du même type, mais pourtant entre les crochets on trouve uniquement une accolade ouverte, ce qui ne représente pas une suite bien formée...

Indice Une solution élégante à ce problème utilise une pile. En parcourant la suite, on push chaque parenthèse ouverte sur la pile, et pour chaque parenthèse fermée on effectue un pop. Si l'opération pop ne fournit pas une parenthèse ouverte du même type que la parenthèse fermée qui l'a déclenchée, c'est que la suite n'est pas bien formée... A la fin la pile doit être vide.

Solution de l'exercice 4

```c
#include <stdio.h>

char pile[100];
const int taille_max = 100;
int taille;

int push(char objet);
int pop(char *p_objet);

int main()
{
    char expression[100];

    scanf("%s", expression);
```

```
15
16      char *p = expression;
17      int ok = 1; // Tout va bien pour l'instant
18      taille = 0; // Pile vide
19
20
21      // p[0] est la meme chose que *p
22      while (p[0]) // p n'est pas "fin de chaine"
23      {
24          if (p[0] == '(' || p[0] == '[' || p[0] == '{')
25          {
26              // push les parenthèses ouvertes
27              push(p[0]);
28          }
29          else
30          {
31              // Parenthèse fermée, faisons un pop
32              char open;
33              if (pop(&open))
34              {
35                  // Il y a bien qqch sur la pile
36                  if (
37                      open == '(' && p[0] != ')' ||
38                      open == '[' && p[0] != ']' ||
39                      open == '{' && p[0] != '}'
40                  )
41                  {
42                      // La parenthèse ouverte sur la pile
43                      // est du mauvais type...
44                      ok = 0;
45                      break;
46                  }
47              }
48              else
49              {
50                  // Pile vide - rien sur la pile qui pourrait
51                  // correspondre à la parenthèse fermée
52                  ok = 0;
53                  break;
54              }
55          }
56          p++;
57      }
```

```
58      // Il faut encore vérifier que la pile est vide!
59      if (ok && !taille)
60      {
61          printf("Suite bien formée!\n");
62      }
63      else
64      {
65          printf("Suite mal formée :(\n");
66      }
67  }
```

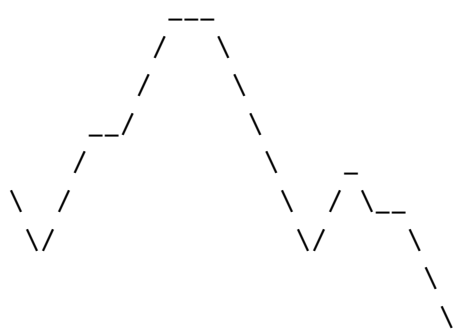
Exo5 (*) Ain't no mountain high enough

Bob est parti faire un tour à vélo. Tous les 100 mètres sa montre connectée enregistre le caractère `/` s'il se trouve dans une montée, le caractère `\` si c'est une descente, et le caractère `_` si la route est plate. On aimerait afficher sur l'écran le profil du parcours de Bob. Souvenez-vous qu'en C le caractère `\` s'écrit `'\\'`.

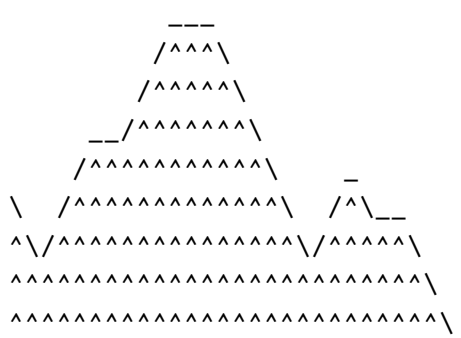
Par exemple, pour la suite

```
\\///__///___\\\\\\//_\__\\\
```

on aimerait afficher :



Bonus : pouvez-vous aussi colorier la montagne ?



Solution de l'exercice 5

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main()
{
    char parcours[100];

    scanf("%s", parcours);
    int len = strlen(parcours);

    int *height = malloc(len * sizeof(int));
    int max_height = 0, current_height = 0;
    char previous = 0;
    for (int i = 0; i < len; i++)
    {
        if (i > 0)
        {
            // Il existe un élément précédent
            if (previous == '/' && parcours[i] == '/' ||
                previous == '/' && parcours[i] == '_')
            {
                // On monte!
                current_height++;
            }
            else if (previous == '_' && parcours[i] == '\\' ||
                     previous == '\\' && parcours[i] == '\\')
            {
                // On descend!
                current_height--;
            }
        }
        if (max_height < current_height)
        {
            max_height = current_height;
        }
        height[i] = current_height;
        previous = parcours[i];
    }

    int displayed = 0; // Combien d'étapes on été affichées

```

```
    // On itère sur les hauteurs possibles
    for (int h = 0; h < len && displayed < len; h++)
    {
        // Il faut décider si on colorie le début de la ligne
        int montagne = 0; // Si c'est 1, alors il faut colorier
        for (int i = 0; i < len; i++)
        {
            if (max_height - height[i] == h)
            {
                // Il faudra dessiner une étape à cette hauteur
                if (parcours[i] == '\\')
                {
                    // On rencontre une descente en premier
                    // donc il faut colorier depuis le début
                    // de la ligne
                    montagne = 1;
                    break;
                }
                else if (parcours[i] == '/')
                {
                    // On rencontre une montée en premier
                    // donc il ne faut pas colorier
                    // depuis le début de la ligne
                    montagne = 0;
                    break;
                }
            }
        }
        // Maintenant on dessine pour de vrai
        for (int i = 0; i < len; i++)
        {
            if (max_height - height[i] == h)
            {
                printf("%c", parcours[i]);
                displayed++;
                if (parcours[i] == '/' || parcours[i] == '\\')
                {
                    // Si on était en train de colorier,
                    // il ne faut plus le faire à partir
                    // de ce point, et vice-versa
                    montagne = 1 - montagne;
                }
            }
```

```
86              else
87              {
88                  // Si montagne, alors on colorie
89                  printf(montagne ? "^" : " ");
90              }
91          }
92          printf("\n");
93      }
94
95      free(height);
96  }
```